МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт Информационных Технологий

Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

«Системный анализ данных СППР»

Студент группы:ИКБО-42-23 Голев С.С.
(Ф. И.О.студента)

Преподаватель Железняк Л.М.
(Ф.И.О. преподавателя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Муравьиный алгоритм относится к роевым методам оптимизации, основанным на моделировании поведения колоний муравьёв при поиске кратчайших путей между источником пищи и гнездом. Основная идея заключается в использовании механизма обмена информацией через феромоны — следы, оставляемые муравьями на маршруте. Эти следы постепенно усиливаются при прохождении по более выгодным путям и испаряются со временем, что обеспечивает баланс между исследованием новых направлений и закреплением найденных решений. В работе рассматривается применение муравьиного алгоритма для решения задачи коммивояжёра. Такой подход позволяет эффективно находить приближенные оптимальные решения в сложных комбинаторных пространствах.

4 МУРАВЬИНЫЙ АЛГОРИТМ

Алгоритм основан на поведении муравьев, и представляет из себя стаю муравьёв перемещающихся на графе для поиска наилучшего пути.

4.1 Цель и задачи практической работы

Целью практической работы является изучение принципов муравьиного алгоритма и его применение для решения задач нахождения кратчайшего пути. В рамках работы необходимо освоить механику работы феромонов, которые помогают найти лучший путь

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Выполнить ручной расчёт одной итерации муравьиного алгоритма для гамильтонова графа;
2. Реализовать муравьиный алгоритм на языке Python для автоматического поиска кратчайшего пути;
3. Применить алгоритм для решения задачи коммивояжёра;
4. Сравнить эффективность результатов ручного расчёта и программной реализации, выявив преимущества алгоритма для задач оптимизации.

4.2 Постановка задачи

В рамках практической работы необходимо реализовать роевый алгоритм вручную и кодово, алгоритм будет проверятся на задаче коммивояжёра где граф состоит из шести вершин.

4.3 Ручной расчёт

Выполним ручной расчёт одной итерации, для гамильтонова графа.

Перемещение по графу основывается на вероятности, с которой выбирается следующая вершина и рассчитывается по формуле:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)}{\sum_{j \in N_i^k} \tau_{ij}^{\alpha}(t)}, \text{если } j \in N_i^k \\ 0, \text{в ином случае} \end{cases} \quad (4.1)$$

Здесь N_i^k представляет множество возможных вершин, связанных с -й вершиной, для k-го муравья. Если для любого -го узла и k-го муравья $N_i^k = \emptyset$, тогда предшественник узла i включается в N_i^k. В этом случае в пути возможны петли. Когда все муравьи построили полный путь от начальной до конечной вершины, удаляются петли в путях, и каждый муравей помечает свой построенный путь, откладывая для каждой дуги феромон в соответствии со следующей формулой.

$$\Delta\tau_{ij}^{k}(t) = \frac{1}{L^k(t)} \quad (4.2)$$

Здесь $L^k(t)$ – длина пути, построенного k-м муравьем в момент времени t.

Таким образом, для каждой дуги графа концентрация феромона определяется следующим образом:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^{k}(t) \quad (4.3)$$

где n_k - число муравьев.

Построим матрицу, которая будет описывать граф для ручного расчёта.

Таблица 4.1 – Значения матрицы

	1	2	3
1		2	3
2	2		1
3	3	1	

Исходное количество муравьев: 3.

Зададим начальные феромоны случайно, но выберем одинаковые значения, τ возьмём для всех рёбер 0.5.

Так как τ одинкова для всех вершин, возьмём перемещение на вторую вершину, далее перемещаемся на третью вершину и возвращаемся к изанчальной.

Путь для первого муравья: $1 \to 2 \to 3 \to 1$

Длинна пути: $2 + 1 + 3 = 6$

Пусть второй и третий муравьи пройдут другим путём.

Путь для второго и третьего муравья: $1 \to 3 \to 2 \to 1$

Длинна пути: $3 + 1 + 2 = 6$

Каждый муравей добавил феромоны на пройденные пути, для каждого длинна равна 6, следовательно $\Delta\tau = 1/6$ и обновленное значение феромонов $\tau = 1/12$.

При следующих итерациях данные шаги будут повторяться и муравьи будут находить наилучшее решение задачи Коммивояжёра.

4.4 Результат работы

Реализуем нахождение кратчайшего пути с помощью муравьиного алгоритма, количество муравьев: 10, количество итераций: 100, выполним реализацию на языке Python. Реализация представлена в приложении Г.

```
(venv) PS C:\Users\semen\Desktop\MIREA\System_data_analysis\Practice4> py .\myravei.py
Кратчайший путь: 1 -> 4 -> 5 -> 6 -> 3 -> 2 -> 1
Длина: 13
```

Рисунок 4.1 – Пример выполнения муравьиного алгоритма

4.5 Результат работы нахождения кратчайшего пути

В ходе практической работы был выполнен ручной расчёт одной итерации муравьиного алгоритма.

Данный расчёт демонстрирует работу муравьиного алгоритма: муравьи находят кратчайший путь, основываясь на ферамонах, которые оставляют другие муравьи прошедшие эти пути.

Далее была выполнена кодовая реализация данного алгоритма, которая позволила:

1. Автоматически находить кратчайший путь;
2. Применять алгоритм для решения задачи коммивояжёра;

В результате работы показано, что муравьиный алгоритм является эффективным методом для нахождения кратчайшего пути и может использоваться в многих практических задачах использующих графы.

ЗАКЛЮЧЕНИЕ

В ходе работы был реализован муравьиный алгоритм для решения задач комбинаторной оптимизации. Эксперименты показали, что данный метод позволяет эффективно находить приближённые оптимальные решения, минимизируя длину маршрута в задаче коммивояжёра. За счёт механизма обновления феромонов алгоритм адаптируется к найденным удачным решениям и постепенно усиливает перспективные направления поиска. Коллективное поведение агентов обеспечивает баланс между исследованием новых путей и использованием накопленного опыта. Полученные результаты подтверждают эффективность и устойчивость муравьиного алгоритма, а также его пригодность для решения широкого круга оптимизационных задач.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Python Software Foundation. Python Documentation — [Электронный ресурс]. URL: https://docs.python.org/3/ (дата обращения: 15.09.2025).

2. Лутц М. Изучаем Python. 5-е изд. / пер. с англ. — Санкт-Петербург: Символ-Плюс, 2019. — 1648 с.

3. Баляев С. А. Объектно-ориентированное программирование. Учебное пособие. — Москва : ФОРУМ, ИНФРА-М, 2020. — 256 с.

4. Гринберг Д. Программирование на Python 3. Подробное руководство. — Москва : Вильямс, 2014. — 832 с.

ПРИЛОЖЕНИЯ

Приложение Г – Код программы “Муравьиный алгоритм”

Приложение Г

Код программы Муравьиный алгоритм

Листинг Г.1 — Основной алгоритм программы

```
import numpy as np
import random

def createGraph(cNodes):
    structure = {}
    step = 1
    for i in range (cNodes):
        structure[str(i + step)] = []

    for i in range (cNodes):
        for j in range (i + 1,cNodes):
            w = np.random.randint(1, 6)

            structure[str(i + step)].append((str(j + step), w))
            structure[str(j + step)].append((str(i + step), w))

    return structure, step

def aco_tsp(graph, startNode, num_ants=10, iterations=100,
alpha=1):
    nodes = list(graph.keys())
    n = len(nodes)

    tau = {i: {j: np.random.uniform(0.1, 1.0) for j, _ in graph[i]}
for i in graph}
    eta = {i: {j: 1 / w for j, w in graph[i]} for i in graph}

    best_length = float('inf')
    best_path = None

    for t in range(iterations):
        paths = []
        lengths = []

        for _ in range(num_ants):
            start = str(startNode)
            unvisited = set(nodes)
            unvisited.remove(start)
            path = [start]
            total_dist = 0
            current = start

            while unvisited:
                neighbors = [n for n, _ in graph[current] if n in
unvisited]
                if not neighbors:
                    break
```

Листинг Г.2 — Продолжение листинга Г.1

```
                probs = []
                denom = sum((tau[current][j] ** alpha) for j in
neighbors)
                for j in neighbors:
                    p = (tau[current][j] ** alpha) / denom
                    probs.append(p)

                next_node = random.choices(neighbors,
weights=probs, k=1)[0]
                dist = next(w for wj, w in graph[current] if wj ==
next_node)
                total_dist += dist
                path.append(next_node)
                unvisited.remove(next_node)
                current = next_node


            if len(path) == n:
                last, first = path[-1], path[0]
                dist = next(w for wj, w in graph[last] if wj ==
first)
                total_dist += dist
                path.append(first)

            # print(' -> '.join(path))
            # print(total_dist)

            paths.append(path)
            lengths.append(total_dist)

            if total_dist < best_length:
                best_length = total_dist
                best_path = path


        for k in range(num_ants):
            path = paths[k]
            length = lengths[k]
            for i in range(len(path) - 1):
                a, b = path[i], path[i + 1]
                tau[a][b] += 1 / length
                tau[b][a] += 1 / length

    return best_path, best_length
```

Листинг Г.3— Продолжение листинга Г.2

```
if __name__ == '__main__':
    cNodes = 6
    structure, start = createGraph(cNodes)

    path, weight = aco_tsp(structure, startNode=start, num_ants =
cNodes)
    print("Кратчайший путь:", ' -> '.join(path))
    print("Длина:", weight)
```